Random Forests and Multilayer Perceptrons in 2D and 5D data

Rodrigo Zamagno Medeiros
Department of Electrical and Computer Engineering, Temple University
tun25387@temple.edu

Introduction: This final paper is an analysis on 2 types of machine learning algorithms, a neural-network based one and a non-neural, implemented after the knowledge acquired in class during the semester. The chosen algorithms are a Random Forest and a Multilayer Perceptrons. These algorithms were trained in a data set of 10,000 data of two-dimension and then its performances were evaluated in a development set with 2,000 data points and also 2D. After the best parameters were arranged on the algorithms, they were put through their final examination using first an evaluation set of 2,000 data points with 2D and then a larger data set of five-dimensions. A training set with 10,0000 data points were used so, later, the performance of the trained algorithm was measured in the training data itself and on new development and evaluation data set with 10,000 data points each. After both algorithms are implemented, they error rates are going to be compared using the concept of "statistical significance" learned in class using a confidence level of 95%.

Random Forest: The Random Forest Classifier is considered one of the best machine learning algorithms to implement. It works, as its name implies, with a large number of decision trees operating together to get a result. A decision tree is a data structure that consists of a root node where the information starts and goes down through its decisions brands, where information about the features are used when choosing which brand to go until it finally hits a leaf node, the end of the path, where the answer is allocated. In this algorithm, each decision tree in the forest will have a prediction and the class is chosen given the prediction that has the most votes. This works good because a big number of unrelated models, the trees, acting as a group is going to exceed any single model. And so, the key for a good implementation of this algorithm is to have decision trees with low correlations between each other, this way they can protect each other from their own classification mistakes. For this final project, it was used an existing implementation of a Random Forest Classifier from the scikit-learn python library.

The implementation consisted of creating a model and using the train.txt set to train the algorithm in a data set with two features and then, evaluating its performance on the training data itself and the dev.txt. As a characteristic of this algorithm, it is not hard to get an error rate of 0% on the training set, but this is due to an overfitting which makes the algorithm not perform so well on other data sets, this way, a search for parameters that could balance this performance was made. This is called hyperparameter tuning, where the settings of an algorithm are adjusted to optimize its performance. On the Random Forest algorithm, the main parameters from the library that were exhaustive searches were the number of estimators (nº of trees), the maximum depth of a single tree, the minimum number of samples necessary to divide an internal node and to be at the leaf node and the maximum number of features that are considered when deciding a split. Since this exhaustive search has a high computation and big time expense, a tool from the same library was used where previously selected parameter are randomly selected and their performance compared. Due to this, the best parameters found may still not be the best ones to optimize the algorithm since not all combinations are performed.

The search for better results worked by using the "Random Search Cross Validation" for better parameters and seeing the performance of the result when training on the whole train data and then predicting the class on the train and development data and analysing the error rate. After many runs I got to a conclusion that more than 2,400 estimators would easily overfit the data and less than 1,000 didn't give so much of good results. Other found parameters were minimum samples for split and leaf of 15 and

2, respectively, and maximum depth of 10. After having narrowed this search on one value for each parameter I began to realize that getting best results would be a matter of luck, since they started being not so different from each other, therefore, I sticked to the lowest error rates that I got and ran the algorithm on the 5D data.

Multilayer Perceptrons: Neural Network algorithms, as their names already implies, are developed using neurons as their basic idea. An input goes through a single neuron, where it is multiplied by a "synaptic" weight and the result leaves it as the output. With a single neuron it's hard to achieve anything, that's why a great number of this little computations working together form what we call a neural network. Also, a bunch of neurons randomly organized wouldn't perform great results, for that reason we can order them in different layers, each with a different numbers of neurons forming a type of neural network called Multilayer Perceptron (MLP). In this case, a feature vector (with 2 or 5 features) enters the initial layer where each neuron computes a value that goes to next layer ando so on until the last layer provides a single output that represents the class assignment. Since a lot of this calculations begin to deal with a certain non-linearity, activations functions are used to set thresholds on the outputs for determining whether the result should be passed ahead or not. Of course that for all of this to work, the network need to learn about the data that is classifying, for this we train the data using other functions like backpropagation in a supervised learning. It works, in a summary, by starting the neural network with random weights and then the features of a known class are inserted and go through the whole network, the result is compared to the expected and the difference is use in some calculations to adjust the parameters on the way back at a certain rate.

For this algorithm I started from the most basic neural-network, to see if my implementation was correct and then began to improve it. I started with a single hidden layer with 2 neurons and ran the algorithm. after correcting bugs issues and making it work, the search for better results started. To make the testings, I split the train set in two: 90% of training and 10% of testing. I realized that when I built more than 4 hidden layers the results started to be not so much accurate anymore, probably due to overfitting. Also, because there were so many data points, my hidden layers were about 50 to 100 neurons each. I also tried different activation functions such as "Rectified Linear Unit", "Sigmoid" and "Hyperbolic Tangent", mixing them between layers. ReLU and Sigmoid showed better results when combined in the same algorithm. After having the model defined, I tried to ran the algorithm to predict the development data, but did not got the results I expected, so I put together again the whole training data and changed the other parameters (learning rate, batch size, momentum and epoch) to try to balance the error rates from the training and the development set. Again, every variation was considered to try to have a good result as the same time the neural network doesn't overtrained itself so other data sets could be rightly classified. Once more, after narrowing the parameters, I calculated that the differences between the error rates were not statistically significant anymore and tried to ran my algorithm on the 5D data, making some adjustments given the new number of features.

Results: After running the implemented algorithms on the given data sets, the results obtained were displayed on the table below. The Random Forest algorithm got better results on the training set due to a little more over training then MLP, even though this situation was trying to be avoided. Using the concept of statistical significance to analyses the results, it can be said that, with a confidence level of 95%, the only results statistically significant difference were from both the training sets. This means that it can not be determined if one algorithm did a better performance than the other give these datas, since their error rate differences weren't big enough to reject the null hypothesis. Relatively, the differences between the error rates found aren't sufficient large in contrast with the number of samples in the experiment.

	2D Data			5D Data		
Algorithm	**Train**	**Dev**	**Eval**	**Train**	**Dev**	**Eval**
Scikit-Learn: Random Forest	5.44%	8.40%	8.65%	33.28%	36.83%	37.00%
Pytorch: Multilayer Perceptrons	8.14%	8.30%	8.05%	36.22%	36.87%	36.82%

Table 1 - Error Rates results of the classification of each data set given the algorithm implemented

Conclusions: Even though it can't be said that one algorithm had clearly a better performance than the other, the lower error rates on the neural-network algorithm show that this algorithm may have a better potential to improve the results if a better hyperparameter search can be made with a more power processing computer. Also, it can be concluded that it can be really difficult to find better parameters without overfitting the training data into the model and to find this best balance can cost a lot of time and processing. Overall, looking at the results obtain, even though they didn't overcame last semester results, they were actually pretty good error rates considering the general results obtained by last semester students. Also, the challenge brings an opportunity not only to try to get better results than other but as well as to try to beat our own algorithms.